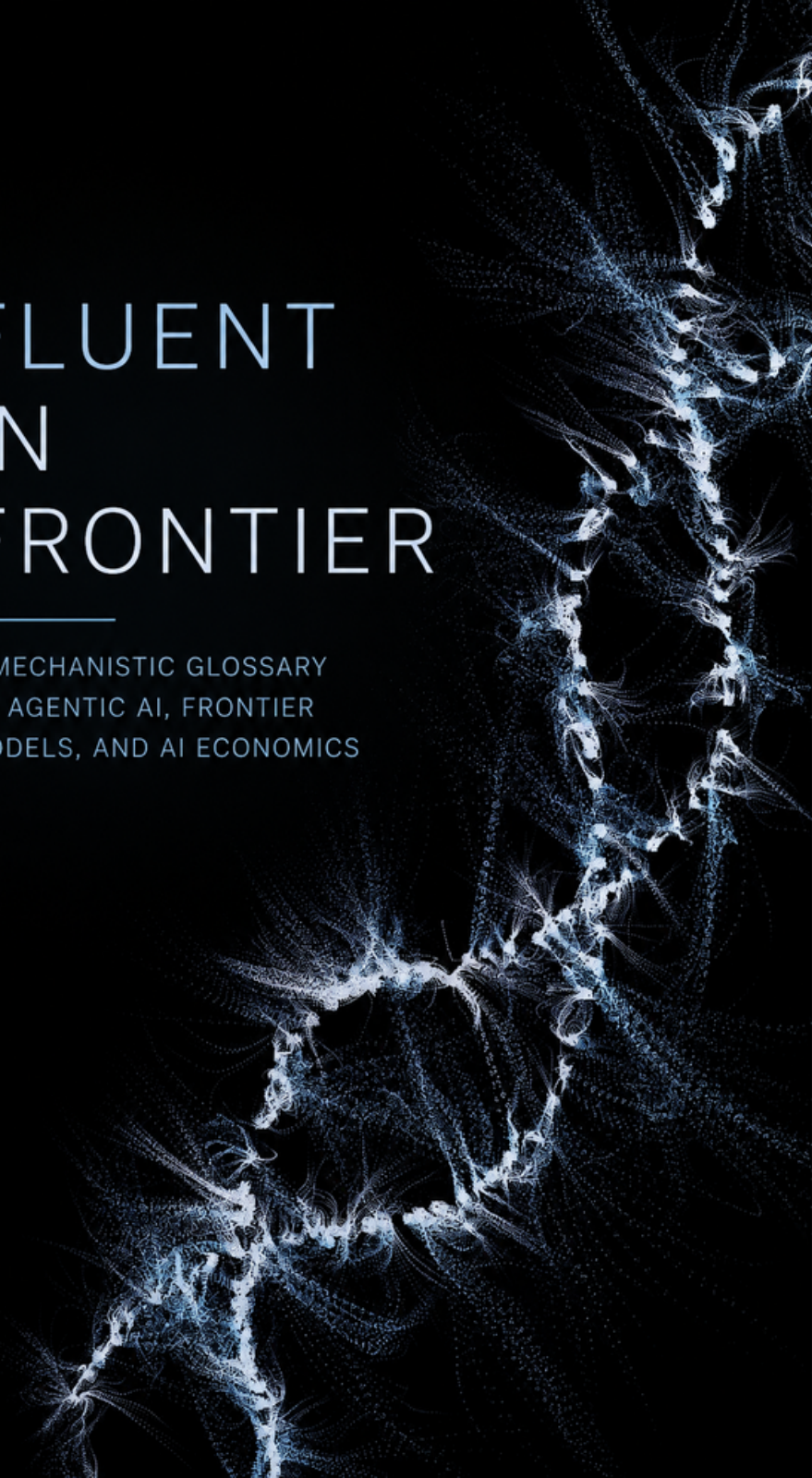


FLUENT IN FRONTIER

A MECHANISTIC GLOSSARY
OF AGENTIC AI, FRONTIER
MODELS, AND AI ECONOMICS



The Fluency Map

A Mechanistic Glossary of Agentic AI, Frontier Models, and AI Economics

Compiled for Gio

The Fluency Map

Front Matter

Part I — Compute & Hardware

Part II — Model Architecture & Representation

Part III — Training

Part IV — Inference & Serving

Part V — Agentic AI

Part VI — Economics & Industry

Part VII — Reasoning Models: A Worked Synthesis

Title Page

Table of Contents

Front Matter

A Mechanistic Glossary of Agentic AI, Frontier Models, and AI Economics

How to use this book

Every entry here follows the same rule: don't just define the term, explain *why it exists and what it causes*. Per Kurt Fischer's Dynamic Skill Theory, a concept becomes "sticky" — automatically recognized, no translation required — only once it's wired into a causal web of neighboring concepts. A definition alone gives you recognition. A mechanism gives you prediction.

So each entry answers three questions:

1. **What is it, mechanically** — what's actually happening under the hood.
2. **Why does it exist** — what problem it solves or constraint it responds to.
3. **What it connects to** — the neighboring concepts that light up when this one is mentioned.

The book is organized into six strands, matching the six webs of causally-linked concepts that show up, interleaved, in almost any frontier-AI conversation: Compute & Hardware, Model Architecture, Training, Inference, Agentic AI, and Economics & Industry. Read one strand to density before moving to the next — parallel-loading all six at once is exactly the working-memory overload that makes podcasts feel "non-sticky."

Part I — Compute & Hardware

The physical substrate. Every architecture decision and every economic argument eventually traces back to a constraint in this section.

GPU (Graphics Processing Unit)

Mechanically, a GPU is thousands of small cores doing simple arithmetic in parallel, versus a CPU's few cores optimized for sequential logic. Neural network training and inference are dominated by matrix multiplication — the same operation repeated millions of times over different data — which parallelizes almost perfectly. That's the whole reason GPUs took over AI: matrix math maps directly onto their architecture. Connects to: TPU, FLOPs, CUDA, NVIDIA, batching.

TPU (Tensor Processing Unit)

Google's custom chip, purpose-built for the exact matrix-multiply-and-accumulate pattern neural nets use, stripping out the general-purpose flexibility a GPU still carries. Because it's narrower in purpose, it can be more efficient per watt and per dollar for training/inference at Google's scale — but it's less flexible for novel architectures. Connects to: GPU, ASIC, custom silicon, hyperscaler.

HBM (High Bandwidth Memory)

Memory stacked vertically and placed physically next to the compute die, rather than on a separate chip connected by a slower bus. The mechanism: model weights and intermediate activations have to move between memory and compute constantly, and if that transfer is slow, the expensive compute cores sit idle waiting for data. HBM exists specifically to reduce that idle time. Connects to: memory bandwidth, memory wall, latency, KV cache.

Bandwidth (memory bandwidth)

The rate at which data can move between memory and compute, measured in GB/s or TB/s. Mechanistically distinct from *compute* (FLOPs) — a chip can have enormous raw compute but still run slowly if it can't feed that compute fast enough. Many modern inference workloads are bandwidth-bound, not compute-bound: the bottleneck isn't how fast the chip can multiply, it's how fast it can retrieve the numbers to multiply. Connects to: HBM, memory wall, inference cost, batching.

Latency vs. Throughput

Latency is time-to-first-response for one request; throughput is total requests processed per unit time across many. The mechanism that creates tension between them: batching multiple requests together raises throughput (more work done per pass over the weights) but raises latency for any individual request (it has to wait for the batch to fill). Every inference-serving decision is a trade-off point on this curve. Connects to: batching, KV cache, inference scaling.

Interconnect (NVLink, InfiniBand)

The wiring between chips, not within them. When a model is too large to fit on one chip, its weights are split (“sharded”) across many, and every forward pass requires those chips to constantly exchange partial results. Interconnect speed determines whether adding more chips actually makes training faster or whether the chips spend their time waiting on each other instead of computing. Connects to: model parallelism, cluster, scaling laws.

FLOPs (Floating Point Operations)

The basic unit of “how much math.” Used both as a hardware spec (FLOPs/second a chip can perform) and as a training-cost unit (total FLOPs consumed to train a model). Because a model's scale is often summarized as “trained with X FLOPs,” this term is the currency that connects hardware capability directly to training economics. Connects to: scaling laws, compute-optimal, capex.

Cluster

A large number of GPUs/TPUs wired together via high-speed

interconnect and treated as one logical machine for training. The mechanism problem clusters solve: a single chip can't hold or compute a frontier model in reasonable time, so the workload is split across thousands of chips that must stay synchronized every training step. Cluster size and interconnect quality — not just chip count — determine effective compute. Connects to: interconnect, parallelism, capex, data center.

Memory Wall

The general phenomenon where compute speed has grown faster than memory bandwidth for decades, so more and more workloads become bottlenecked by data movement rather than arithmetic. This is why HBM, KV cache optimization, and quantization all exist — they're different attacks on the same underlying wall. Connects to: HBM, bandwidth, quantization, KV cache.

Utilization

The percentage of a chip's theoretical peak FLOPs actually being used during a real workload, as opposed to sitting idle waiting on memory or network. Frontier labs obsess over this number because a cluster running at 30% utilization effectively costs three times as much per unit of real training progress as one running at 90%. Connects to: bandwidth, interconnect, capex, unit economics.

Fab / TSMC / Advanced Packaging (CoWoS)

The fab is the physical factory that manufactures chips; TSMC dominates the highest-end fabrication process nodes that NVIDIA and others depend on. "Advanced packaging" (like CoWoS) refers to the process of stacking compute dies and HBM together physically — a separate bottleneck from fabrication itself, since even if enough chips can be *etched*, not enough can be *packaged* into finished products. This is why chip shortages can persist even after fab capacity increases. Connects to: GPU, HBM, export controls, supply chain.

Power & Cooling

Data centers are increasingly constrained not by chip supply but by the electrical grid's ability to deliver power and by the physical ability

to dissipate the heat that power becomes. This is why frontier labs and hyperscalers now sign nuclear and gas power deals directly — compute growth has become a power-generation problem as much as a chip problem. Connects to: capex, data center, hyperscaler.

Part II — Model Architecture & Representation

How raw compute becomes a system that represents and manipulates meaning.

Transformer

The dominant neural network architecture, built around the mechanism of *attention*: instead of processing a sequence word-by-word in strict order (like older recurrent networks), it lets every position in the input look at every other position simultaneously and weigh how relevant each one is. This parallelism is precisely why transformers scale so well on GPUs — the “look at everything at once” operation is itself a giant matrix multiplication. Connects to: attention, embedding, GPU, parameters.

Attention (Self-Attention, Multi-Head)

The core mechanism: for every token, the model computes a “query,” and compares it against the “keys” of every other token to produce weights describing how much each other token should influence this one’s representation. Multi-head attention runs several of these comparison processes in parallel, each potentially capturing a different kind of relationship (grammatical, semantic, positional). This is what lets a model figure out that “it” refers to a specific noun three sentences earlier. Connects to: transformer, context window, embedding.

Embedding

The conversion of a discrete symbol (a word, subword, or token) into a vector of numbers — a point in high-dimensional space — where geometric closeness corresponds to semantic closeness. This is the

mechanism that lets a neural network, which only does arithmetic, operate on language at all: everything becomes numbers whose relationships encode meaning. Connects to: vector, token, tokenization.

Vector

An ordered list of numbers representing a position in a high-dimensional space. In AI, everything — words, images, entire sentences, retrieved documents — eventually becomes a vector so that “similarity” can be computed mathematically as distance or angle between two vectors. This is the substrate that makes embeddings, retrieval, and attention all interoperable. Connects to: embedding, RAG, vector database.

Token / Tokenization

Text is split into chunks (tokens) — often subwords, not full words — before being fed to the model, because a fixed, finite vocabulary of tokens is needed to map text onto embeddings. Mechanistically, this is *why* models are sometimes bad at character-level tasks (like counting letters): they don’t see letters, they see token IDs, and a word might be a single opaque token or a fragment of one, unpredictably. Connects to: embedding, context window, token economics.

Context Window

The maximum number of tokens a model can attend to at once — its working memory. Mechanistically bounded because attention’s compute cost grows roughly with the square of sequence length (every token compares against every other token), so a longer window is dramatically more expensive to compute and to store (see KV cache), not just linearly more expensive. This quadratic cost is the central constraint driving research into more efficient attention variants. Connects to: attention, KV cache, agentic loop, context engineering.

Parameters / Weights

The numbers inside a model that get adjusted during training — the actual “knowledge” is encoded entirely as the specific values of these numbers, not as any explicit rules. “Parameter count” (7B, 70B, 400B)

is a rough proxy for model capacity, but it conflates two very different things once mixture-of-experts architectures are involved (see MoE): total parameters stored vs. parameters actually used per token. Connects to: MoE, training, scaling laws.

Positional Encoding

Because attention compares all tokens to all tokens simultaneously, it has no inherent sense of *order* — “dog bites man” and “man bites dog” would look identical without it. Positional encoding injects information about each token’s position into its representation so the model can distinguish sequence order. Connects to: attention, transformer, context window.

Layers / Depth

A transformer is a stack of repeated attention-and-processing blocks; each layer refines the representation produced by the one before it. Roughly, early layers tend to capture more surface-level patterns and later layers more abstract ones, though this is an emergent property rather than an explicit design choice. Depth (number of layers) is one axis of scaling, distinct from width (size of each layer). Connects to: parameters, scaling laws, transformer.

MoE (Mixture of Experts)

Instead of every parameter in the model processing every token (a “dense” model), an MoE model contains many parallel sub-networks (“experts”), and a small router network decides, per token, which handful of experts to activate. The mechanism payoff: the model can have a huge total parameter count (more stored knowledge) while only activating a small fraction of those parameters for any given token (keeping inference cost close to a much smaller dense model). This is why “active parameters” and “total parameters” are reported separately for MoE models. Connects to: parameters, routing, inference cost.

Dense vs. Sparse Models

A dense model uses all its parameters for every input; a sparse model (MoE being the dominant example) activates only a subset. The trade-

off is that sparse models are more compute-efficient per parameter but harder to train stably (the router itself has to learn a good policy) and less memory-efficient (you still have to store all the experts even if you don't compute with all of them). Connects to: MoE, inference cost, HBM.

Active vs. Total Parameters

For MoE models specifically: total parameters is the full stored model size (determines memory/HBM requirements); active parameters is how many are actually used to process a given token (determines compute cost and, roughly, inference latency). A model can have ten times more total than active parameters — meaning its capability scales like a huge model while its per-token cost scales like a much smaller one. Connects to: MoE, inference cost, capex.

Part III — Training

How a model with random initial weights becomes a model with useful ones.

Pretraining

The first, most compute-intensive phase: the model is trained on a massive corpus of text (or other data) with a simple objective — usually predicting the next token given the preceding ones. Mechanistically, this simple objective, applied at massive scale, is what forces the model to implicitly learn grammar, facts, reasoning patterns, and style, because accurately predicting the next token requires an internal model of all of those things. Connects to: scaling laws, tokens, base model.

Fine-tuning

A second training phase applied *after* pretraining, on a smaller, more targeted dataset, to shift the model’s behavior toward a specific format or task (e.g., following instructions, adopting a persona, excelling at code). Mechanistically it’s the same gradient-descent process as pretraining, just with different (smaller, curated) data and usually a lower learning rate, so it nudges the existing weights rather than re-learning from scratch. Connects to: post-training, RLHF, base model.

Post-training

The umbrella term for everything done to a model *after* pretraining — fine-tuning, RLHF, DPO, safety tuning — that turns a raw “base model” (good at predicting plausible text) into a “chat model” (good at being useful and following intent). This is where most of the *behavioral* character of a released model actually gets shaped, even though pretraining consumes most of the compute. Connects to: fine-tuning, RLHF, alignment.

RLHF (Reinforcement Learning from Human Feedback)

A post-training technique: humans rank multiple model outputs by quality/preference, that ranking data trains a separate “reward model” to predict human preference, and then the main model is updated via reinforcement learning to produce outputs the reward model scores highly. Mechanistically it’s a way to optimize for a fuzzy, hard-to-write-down objective (“be helpful, honest, and harmless”) by proxy, since you can’t write a mathematical loss function for “good” directly. Connects to: reward model, post-training, alignment tax, DPO.

RLAIF / RLVR

Variants of RLHF where the feedback signal comes from an AI model’s judgment (RLAIF) or from an automatically verifiable reward like “did the code pass its unit tests” or “was the math answer correct” (RLVR — reinforcement learning from verifiable rewards) instead of human raters. RLVR in particular is the mechanism underlying most modern “reasoning models” — because math and code have checkable ground truth, the model can be rewarded directly for arriving at correct answers, reinforcing whatever internal reasoning process got it there. Connects to: reasoning models, RLHF, reward model.

DPO (Direct Preference Optimization)

A more recent alternative to RLHF that skips training a separate reward model: it directly adjusts the main model’s weights so that preferred outputs become more probable and rejected ones less probable, using the preference-pair data straight away. Mechanistically simpler and more stable to train than full RLHF, which is why many open-weight releases use it instead. Connects to: RLHF, post-training.

Reward Model

A separate, smaller model trained to take an output and predict a scalar score for “how good is this,” based on human preference data. It exists because reinforcement learning needs a reward signal at every step, and no such signal exists in raw form — the reward model manufactures one, standing in for a human judge at massive scale

during RL training. Connects to: RLHF, RLVR.

Synthetic Data

Training data generated by a model rather than collected from humans/the world — e.g., having a strong model generate worked math solutions, or generate variations of existing examples. Mechanistically important for two reasons: (1) high-quality human data is finite and increasingly exhausted at the frontier, and (2) synthetic data can be filtered or verified (e.g., keep only the math solutions that check out), producing cleaner signal than noisy human-generated text. This is central to the “data flywheel” argument in AI economics — labs with strong models can bootstrap better training data for the next model. Connects to: distillation, RLVR, data moat.

Distillation

Training a smaller “student” model to mimic the outputs (or internal probability distributions) of a larger “teacher” model, transferring much of the teacher’s capability into a cheaper package. Mechanistically effective because the teacher’s full probability distribution over possible next tokens carries more information than a single correct answer would — it implicitly reveals what the teacher considers plausible alternatives, which is a richer training signal than raw data. Connects to: synthetic data, inference cost, frontier lab.

Scaling Laws

Empirically discovered relationships showing that model performance improves predictably (as a power law) with increases in three things: parameter count, data volume, and compute — as long as they’re scaled together in roughly the right ratio. This turned frontier AI development from trial-and-error into something closer to engineering: labs can predict, before spending tens of millions of dollars, roughly how good a model of a given size/data/compute budget will be. Connects to: compute-optimal, FLOPs, capex.

Compute-Optimal (Chinchilla)

A specific, influential scaling-law finding: for a fixed compute budget, there’s an optimal balance between model size and training data

volume, and many earlier models were “overparameterized” relative to how much data they’d seen — meaning a smaller model trained on more data would have performed better for the same compute cost. This reshaped how labs allocate training budgets. Connects to: scaling laws, FLOPs, pretraining.

Alignment / Alignment Tax

Alignment is the general goal of making a model’s behavior match human intent and values, not just its raw predictive capability. “Alignment tax” refers to the observed phenomenon that safety/alignment tuning sometimes reduces raw capability or makes a model more conservative/refusal-prone — a real trade-off labs have to manage, not just a hypothetical concern. Connects to: RLHF, post-training, guardrails.

Overfitting / Data Contamination

Overfitting: a model that has essentially memorized its training data rather than learned generalizable patterns, performing well on seen examples but poorly on new ones. Data contamination is a specific frontier-AI version of this problem: benchmark test questions leaking into training data (often unintentionally, since training corpora scrape the internet where benchmarks are discussed), making a model’s benchmark scores look better than its real-world capability. Connects to: pretraining, benchmarks, evals.

Base Model vs. Chat/Instruct Model

The base model is the raw output of pretraining — it’s good at continuing text plausibly but has no notion of “being an assistant” and will just as happily continue a question with more questions as it will answer it. The chat/instruct model is that same base model after post-training has shaped it specifically to interpret inputs as instructions and produce helpful, bounded responses. This distinction matters because API access to a “base model” behaves noticeably differently from the chat product built on top of it. Connects to: pretraining, post-training, RLHF.

Part IV — Inference & Serving

What happens after training, every time the model is actually used.

Inference

The process of running a trained (fixed) model forward to produce an output, as opposed to training, where weights are being updated. Mechanistically simpler than training (no gradient computation, no backward pass) but happens vastly more often — a model is trained once and then run for inference billions of times, which is why inference cost, not training cost, dominates a frontier lab’s ongoing compute spend at scale. Connects to: KV cache, batching, inference cost.

KV Cache

During autoregressive generation (producing one token at a time), the model would naively have to recompute attention over the entire preceding sequence for every new token — wasteful, since most of that computation doesn’t change. The KV cache stores the “key” and “value” vectors computed for earlier tokens so they can be reused rather than recomputed. Mechanistically, this cache grows with context length and is a major consumer of HBM during long-context or multi-user serving, which is why context window and memory cost are directly linked. Connects to: context window, HBM, memory wall.

Quantization

Reducing the numerical precision used to store model weights (e.g., from 16-bit to 8-bit or 4-bit numbers), trading a small amount of accuracy for large reductions in memory footprint and often faster computation. Mechanistically it works because neural networks are surprisingly tolerant of imprecise weights — the aggregate behavior across billions of parameters is robust to individual numbers being slightly less exact. This is one of the primary levers for cutting

inference cost. Connects to: memory wall, inference cost, HBM.

Batching

Grouping multiple inference requests together so they're processed in a single pass through the model rather than one at a time, dramatically improving GPU utilization since the same weights get reused across many requests per memory-load. The mechanism trade-off: batching improves throughput but increases latency for individual requests (see Latency vs. Throughput), because a request has to wait for the batch to be assembled. Connects to: throughput, GPU, utilization.

Speculative Decoding

A latency-reduction trick: a small, fast “draft” model generates several candidate next-tokens ahead of time, and the large model verifies (in a single parallel pass) whether it would have produced the same tokens, accepting the correct ones and only falling back to token-by-token generation where the draft was wrong. Mechanistically this exploits the fact that verifying is cheaper than generating, since verification can be parallelized across the whole draft sequence at once. Connects to: latency, inference cost, distillation.

Routing

In an MoE model, routing is the mechanism by which a small learned network decides which experts should process each token. More broadly in industry usage, “routing” also refers to sending a user’s request to different models entirely — a cheap fast model for simple queries, an expensive frontier model for hard ones — as a cost-optimization strategy. Connects to: MoE, inference cost, unit economics.

Reasoning Tokens / Chain-of-Thought

Reasoning models generate intermediate tokens — a visible or hidden “thinking” process — before producing a final answer, rather than jumping straight to a response. Mechanistically, this works because transformers do a fixed amount of computation per output token; by generating extra intermediate tokens, the model effectively gets to

spend more total compute on a hard problem, using its own prior outputs as a scratchpad it can condition on. This is why reasoning models are slower and more expensive per query — they’re literally doing more forward passes before answering. Connects to: test-time compute, RLVR, inference scaling.

Test-Time Compute / Inference Scaling

The discovery that model performance can be improved not just by scaling training (bigger model, more data) but by scaling how much computation is spent *at inference time* on a given query — through longer chains of reasoning, multiple sampled attempts with a verifier picking the best, or search over possible solution paths. This is mechanistically a second, independent scaling axis from pretraining scaling laws, and it’s the direct cause of the industry narrative shift from “just make bigger models” toward “make models that think longer.” Connects to: reasoning tokens, scaling laws, RLVR.

Sampling / Temperature

When generating text, the model produces a probability distribution over possible next tokens, not a single deterministic answer; “sampling” is the process of actually picking one token from that distribution. Temperature controls how sharply peaked that sampling is — low temperature makes the model nearly always pick the highest-probability token (more deterministic, more repetitive), high temperature flattens the distribution (more varied, more error-prone). Connects to: token, inference, hallucination.

Self-Consistency / Best-of-N

A test-time technique: generate multiple independent answers to the same problem (via sampling) and either take a majority vote or use a separate verifier/reward model to pick the best one. Mechanistically this works because errors in generation are often somewhat random, while correct reasoning paths tend to converge on the same answer — so aggregating across samples filters out noise. This is one of the concrete techniques underlying “test-time compute.” Connects to: test-time compute, reward model, verifier.

Part V — Agentic AI

Its own causal web, distinct from base-model vocabulary: this is about models acting over time, using tools, and managing state, not just answering single questions.

Agent

Loosely, a model wrapped in a loop that lets it take actions (not just produce text), observe the results of those actions, and decide what to do next — repeated until a task is complete. Mechanistically, “agent” isn’t a different kind of model, it’s an architecture pattern built *around* a model: the same underlying LLM call is invoked repeatedly, each time with an updated context that includes what happened since the last call. Connects to: agentic loop, tool use, orchestration.

Tool Use / Function Calling

The mechanism by which a model can trigger an external action — calling an API, running code, querying a database — rather than only producing text. Concretely, the model is given a list of available “tools” (with names, descriptions, and expected inputs) in its context, and instead of answering directly it can output a structured request to invoke one; the actual execution happens outside the model, and the result is fed back into its context for the next step. This is the single mechanism that turns a text-predictor into something that can *do* things. Connects to: agent, MCP, agentic loop.

MCP (Model Context Protocol)

A standardized protocol for connecting models to external tools and data sources, so that any tool built to the MCP spec can be plugged into any MCP-compatible model/application without custom integration code for each pairing. Mechanistically it solves an $N \times M$ integration problem: without a standard, every AI application needed bespoke code for every tool it wanted to support; MCP turns that into

an $N + M$ problem, where tools and applications each only need to speak the protocol once. Connects to: tool use, orchestration, agent.

Orchestration

The logic that manages a multi-step or multi-agent workflow: deciding what runs when, in what order, with what shared state, and how results get passed between steps or between different agents. Mechanistically distinct from a single model call — orchestration is the surrounding code/system (often not itself an LLM call at every step) that decides *whether* to call the model again, call a different model, call a tool, or terminate. Poor orchestration is a common practical failure point even when the underlying model is capable. Connects to: agent, multi-agent systems, agentic loop.

Context Engineering

The practice of deliberately curating exactly what information goes into a model's context window at each step of an agentic task — because context windows are finite and because irrelevant or excessive context measurably degrades performance (not just cost). Mechanistically this responds to a real limitation: attention cost grows with context length, and models empirically get worse at using information buried in the middle of a long, cluttered context (“lost in the middle”). This is why agent frameworks spend significant effort on summarization, retrieval filtering, and pruning stale information rather than just stuffing everything in. Connects to: context window, RAG, agentic loop.

RAG (Retrieval-Augmented Generation)

Instead of relying solely on knowledge baked into the model's weights during training, RAG retrieves relevant documents/passages from an external source (typically via vector similarity search) at query time and inserts them into the context before generation. Mechanistically this exists because a model's weights are frozen after training — RAG is the primary way to give a model access to information that's newer, private, or too voluminous to have been trained on, without retraining. Connects to: vector, embedding, context engineering, hallucination.

Vector Database

A database optimized for storing embeddings and performing fast similarity search (finding the nearest vectors to a query vector) at scale, which is the retrieval mechanism underneath most RAG systems. Mechanistically it trades exact lookup (like a normal database's indexed key search) for approximate *semantic* lookup (find things that mean something similar), which is only possible because embeddings place similar meanings near each other geometrically. Connects to: vector, embedding, RAG.

Memory (Short-Term vs. Long-Term)

Short-term memory in an agent is just whatever's in the current context window — it vanishes once the session ends or the context is cleared. Long-term memory is information deliberately persisted outside the context window (in a database, file, or vector store) and selectively retrieved back into context in future sessions. This distinction matters mechanistically because it's the difference between a model “knowing” something because it's actively attending to it right now, versus a system that can *choose* to re-surface relevant past information on demand. Connects to: context window, RAG, state management.

Agentic Loop

The repeating cycle underlying most agent architectures: observe (get current state/context) → think (model reasons about what to do) → act (call a tool or produce output) → observe the result → repeat, until a stopping condition is met. Mechanistically it's the same as ReAct-style prompting, generalized into a system architecture — the model doesn't do the whole task in one shot, it does one step, sees what happened, and re-plans. This is also the mechanism by which small per-step error rates compound into large task-failure rates over long-horizon tasks. Connects to: agent, orchestration, tool use.

ReAct (Reason + Act)

A specific, influential pattern for structuring the agentic loop: at each step, the model is prompted to explicitly produce a reasoning trace (“I should check X because...”) *before* producing an action, interleaving thought and action rather than jumping straight to acting.

Mechanistically this improves reliability because it forces the model to externalize its plan into tokens it can then condition on, similar to how chain-of-thought improves single-turn reasoning. Connects to: agentic loop, reasoning tokens, tool use.

Multi-Agent Systems

Architectures where multiple separate model instances (often with different prompts, tools, or even different underlying models) collaborate on a task, each handling a sub-piece and passing results to each other, rather than one model handling everything.

Mechanistically this exists partly to work around context-window and attention-degradation limits (each sub-agent has a smaller, more focused context) and partly to allow specialization (a “coding agent” and a “research agent” with different tool access). The trade-off is added orchestration complexity and new failure modes around handoff and coordination. Connects to: orchestration, subagents, handoff.

Subagents / Handoff

A subagent is an agent spawned by another (often orchestrating) agent to handle a specific bounded piece of a larger task, typically with its own separate context window. “Handoff” is the mechanism of transferring control (and relevant state/context) from one agent to another mid-task. This pattern exists because giving one agent an ever-growing, ever-more-cluttered context to handle an entire complex task degrades performance — subagents let each piece of work start with a clean, focused context. Connects to: multi-agent systems, context engineering, orchestration.

Autonomy Level

A rough spectrum describing how much of a task loop happens without human review: from “suggests one action, human approves each step” up to “operates for extended periods, taking many actions and only surfacing for review at the end or on failure.” Higher autonomy multiplies both usefulness (less human bottleneck) and risk (errors compound further before being caught), which is why autonomy level is often treated as a deliberate, adjustable dial rather than a fixed property of a system. Connects to: guardrails, agentic loop, evals.

Guardrails

Constraints placed around a model or agent's possible actions — input filtering, output filtering, restricted tool permissions, required human approval for certain actions — that exist independently of the model's own judgment. Mechanistically important because model behavior is probabilistic and imperfect even after alignment training; guardrails are a systems-level backstop rather than relying solely on the model “deciding” not to do something harmful. Connects to: alignment, autonomy level, tool use.

Evals / Evaluation Harnesses

Structured tests used to measure a model or agent's performance on specific tasks, ranging from simple benchmark question sets to complex simulated environments that score whether an agent successfully completed a multi-step task. Mechanistically distinct from benchmarks used during training — evals are how labs and companies decide whether a new model/agent version is actually better before shipping it, and increasingly, agentic evals have to measure success over long, multi-step trajectories rather than single answers, which is a much harder measurement problem. Connects to: benchmarks, data contamination, autonomy level.

Hallucination

A model producing output that's fluent and confident but factually wrong or unsupported, arising mechanistically because the model is fundamentally predicting plausible-sounding text, not retrieving verified facts — it has no built-in mechanism to distinguish “this pattern is common in my training data” from “this is true.” RAG, verifier models, and tool use (letting the model check a fact rather than recall it) are the main mechanisms used to reduce, though not eliminate, hallucination. Connects to: RAG, grounding, verifier.

Grounding

Connecting a model's outputs to a verifiable external source of truth — a document, a database, a tool result — rather than letting it rely purely on internalized weights. Mechanistically this is the general category that RAG, tool use, and citation-based generation all fall into: different specific techniques for reducing the gap between “what the

model says” and “what’s actually true or current.” Connects to: RAG, hallucination, tool use.

State Management

The practice of tracking what has happened so far in a multi-step agentic task — which steps are complete, what data has been gathered, what the current plan is — persistently, so it survives across multiple model calls (each of which is, mechanistically, a fresh, stateless invocation of the model). Because the underlying model itself has no memory between calls, everything about continuity in an agent has to be engineered explicitly into the surrounding system. Connects to: memory, context engineering, orchestration.

Part VI — Economics & Industry

Why the hardware and architecture facts above translate into the business narratives you hear on investor podcasts.

Capex vs. Opex

Capex (capital expenditure) is upfront spending on long-lived assets — building data centers, buying GPU clusters; opex (operating expenditure) is ongoing running costs — power, staffing, model API calls. Mechanistically this distinction matters financially because capex is depreciated over years on a balance sheet (affecting how profitable a company *looks* in the short term) while opex hits the income statement immediately, so the current wave of massive AI capex spending doesn't show up as an equivalent hit to reported near-term profit, which is a recurring point of debate among investors. Connects to: hyperscaler, cluster, depreciation.

Hyperscaler

The small group of companies (Microsoft, Google, Amazon, Meta, and a few others) operating cloud infrastructure at massive scale, who are both major AI infrastructure *providers* (renting out compute) and major AI infrastructure *consumers* (training their own frontier models). This dual role is mechanistically important: hyperscalers' capex spending is simultaneously a bet on their own future model capabilities and a bet on external demand for renting that compute to others. Connects to: capex, frontier lab, cluster.

Frontier Lab

Organizations (OpenAI, Anthropic, Google DeepMind, and a handful of others) actively training models at or near the current capability ceiling, as opposed to companies building products on top of existing models. Mechanistically distinguished by the scale of compute investment required to be “frontier” at all — the capital requirement

itself acts as a barrier to entry, which is central to arguments about market concentration in the industry. Connects to: hyperscaler, scaling laws, compute moat.

Foundation Model

A large pretrained model (like GPT, Claude, or Gemini) designed to be broadly capable and then adapted — via fine-tuning, prompting, or RAG — to many different downstream applications, rather than being trained for one narrow task. The term emphasizes the economic pattern: enormous upfront training cost, amortized across a huge number of downstream uses built on top of it. Connects to: pretraining, base model, API.

Open Weights vs. Closed Models

Open-weight models release the trained parameters publicly (anyone can download and run them, e.g., via Meta's Llama or various Chinese labs' releases); closed models are only accessible via API, with the weights kept private. Mechanistically this affects the business model directly: closed labs monetize via per-token API pricing and can control/update behavior centrally, while open-weight releases monetize indirectly (ecosystem/platform strategy, enterprise services, competitive pressure on rivals' pricing) since the weights themselves, once released, can't be taken back. Connects to: API, token economics, distillation.

API vs. Subscription (Business Model)

API pricing charges per unit of usage (typically per token processed); subscription pricing charges a flat recurring fee for access regardless of usage volume. Mechanistically these create very different cost/margin dynamics for the provider: API revenue scales directly with compute cost incurred (more usage = more revenue but also more inference cost), while subscription revenue is fixed against variable underlying cost, meaning heavy users are more expensive to serve than their subscription fee covers, and light users cross-subsidize them. Connects to: token economics, inference cost, unit economics.

Token Economics

The unit economics of running a model, expressed per token: cost to generate a token (driven by inference cost — chip cost, power, utilization) versus price charged per token (set by the provider) versus how many tokens a given task consumes (driven by context window size, reasoning-token usage, and agentic loop length). Mechanistically, this is why reasoning models and long agentic workflows are more expensive than simple chat — they consume dramatically more tokens per task, even at the same per-token price. Connects to: inference cost, reasoning tokens, API.

Inference Cost / Marginal Cost

The cost incurred each time a model actually processes a request, as opposed to the (already sunk) cost of training it. Mechanistically this is the number that determines whether an AI product is profitable per-use: as usage scales, training cost gets amortized toward zero per query, but inference cost doesn't — it's incurred fresh on every single request, which is why efficiency techniques (quantization, distillation, MoE, speculative decoding) are economically, not just technically, important. Connects to: quantization, MoE, unit economics.

Unit Economics

Whether a business makes or loses money on each individual unit of what it sells — in AI, typically “cost to serve one query/token” vs. “revenue from one query/token.” Mechanistically distinct from overall company profitability, because a company can have terrible unit economics (losing money on every single query) while still growing revenue rapidly, betting that future efficiency gains or pricing power will flip the unit economics positive before capital runs out. This is a central point of skepticism in AI-economics commentary. Connects to: inference cost, capex, gross margin.

Compute Moat / Data Moat / Distribution Moat

Three distinct competitive-advantage arguments in AI: a *compute moat* is advantage from having more/cheaper access to training and inference infrastructure than competitors; a *data moat* is advantage from proprietary or higher-quality training data others can't access; a *distribution moat* is advantage from already having the user

relationship (e.g., a hyperscaler bundling AI into products billions already use), independent of whether the underlying model is best-in-class. Mechanistically these moats can each erode differently — compute moats shrink as chips get cheaper/more available, data moats shrink as synthetic data improves, distribution moats are the stickiest because they depend on switching costs, not technical capability. Connects to: frontier lab, synthetic data, capex.

Talent Acquisition (as M&A strategy)

A pattern where a company “acquires” a startup primarily to obtain its research team, sometimes without keeping the underlying product running, structured as licensing deals or direct hires rather than traditional acquisitions. Mechanistically this reflects that in frontier AI, the scarce resource is often specific research talent and know-how (which is hard to hire piecemeal) more than any particular codebase or product. Connects to: frontier lab, compute moat.

GPU Supply / Export Controls

Government restrictions (notably U.S. export controls) limiting which countries or entities can purchase the most advanced AI chips, aimed at slowing rival nations’ frontier-AI progress. Mechanistically this creates real supply segmentation — different chip tiers get designed and sold specifically to comply with different countries’ restriction levels — and drives affected countries to invest in domestic chip fabrication as a workaround, which is itself a multi-year undertaking given fab complexity. Connects to: fab/TSMC, GPU, geopolitics.

Circular Deals / Vendor Financing

A pattern of scrutiny in current AI-infrastructure investment where a company (e.g., a chipmaker) invests in or extends credit to a customer (e.g., an AI lab), which then uses that capital to buy the investor’s own products — inflating both companies’ apparent revenue/growth without new external capital actually entering the system. Mechanistically this matters because it can make demand look more organic and durable than it is, which is a recurring point of debate when evaluating whether current AI infrastructure spending reflects real end-demand. Connects to: capex, hyperscaler.

Depreciation (of AI hardware)

The accounting practice of spreading a capital asset's cost over its useful life rather than expensing it all at once — for GPUs, the assumed “useful life” (often 3-6 years) is a major variable, because GPUs may become economically obsolete (outcompeted by newer, more efficient chips) well before that assumed lifespan ends. Mechanistically, if real useful life is shorter than the assumed depreciation schedule, reported profitability across the industry is currently overstated relative to true economic cost — a frequently raised concern in AI-capex skepticism. Connects to: capex, hyperscaler, unit economics.

Part VII — Reasoning Models: A Worked Synthesis

This section exists to demonstrate the goal of the whole book: watching several entries above combine into one compressed chunk.

Consider the sentence from your example: “**Reasoning models shift compute from training toward inference.**”

Unpacked, using entries from every strand in this book:

A traditional model spends its compute budget almost entirely during **pretraining** (Part III) — huge upfront cost, governed by **scaling laws**, to produce a fixed set of **weights**. Once trained, running it (**inference**, Part IV) is comparatively cheap: one forward pass per token.

A **reasoning model** changes this balance. Through **RLVR** post-training (Part III) — reinforcement learning against verifiable rewards like “did the code pass” or “was the math right” — the model learns to generate extra **reasoning tokens** (Part IV) before answering: a scratchpad of intermediate steps that lets it effectively spend more forward passes, and therefore more compute, on any single hard query. This is **test-time compute** (Part IV): a second scaling axis, independent of pretraining scale.

The economic consequence (Part VI): **token economics** shift. A query that used to cost a fixed, small number of tokens might now cost ten or a hundred times as many, because the model is “thinking” before answering. **Inference cost** — previously a rounding error next to training capex — becomes a first-order line item, especially at the usage volumes agentic systems (Part V) generate, where a single task might trigger dozens of reasoning-heavy model calls inside an **agentic loop**.

That’s the whole mechanism behind one sentence that used to require twelve unfamiliar words to parse. Once each piece is a chunk rather than a word, the sentence collapses into something closer to hearing

“cars use gasoline” — obvious, fast, requiring no translation.

This is the target state for every entry in this book: not memorized definitions, but pre-built chunks ready to combine the instant they’re triggered by conversation.